NATIONAL BUREAU OF STANDARDS REPORT

2047

SWAC CODING GUIDE

by

Ruth B. Horgan

NBS

U. S. DEPARTMENT OF COMMERCE
NATIONAL BUREAU OF STANDARDS

1101-30-5103 November 4, 1952 2047

SWAC CODING GUIDE*

Ruth B. Horgan

National Bureau of Standards

*The preparation of this paper was sponsored (in part) by the Office of Scientific Research, USAF

TABLE OF CONTENTS

The material in this handbook is valid as of the date of issue.

In all cases of disagreement, the information of most recent date supersedes all information previously issued. Additional data will be supplied as the need develops.

Suggestions of modifications and additional material for inclusion will be welcomed.

MEANINGS OF THE ADDRESSES FOR THE COMMANDS USED BY THE SWAC.

COMMAND	α	β	γ	δ	F	Eq.**
ADD	ADDRESS OF AUGEND	ADDRESS OF ADDEND	ADDRESS OF SUM	ADDRESS OF NEXT COMMAND IF OVERFLOW	a	4
SPECIAL ADD	ADDRESS OF AUGEND	ADDRESS OF ADDEND	ADDRESS OF SUM	ADDRESS OF NEXT COMMAND	b	5
SUBTRACT	ADDRESS OF MINUEND	ADDRESS OF SUBTRAHEND	ADDRESS OF DIFFERENCE	ADDRESS OF NEXT COMMAND IF OVERFLOW	s	6
SPECIAL SUBTRACT	ADDRESS OF MINUEND	ADDRESS OF SUBTRAHEND	ADDRESS OF DIFFERENCE	ADDRESS OF NEXT COMMAND	t	7
MULTIPLY	ADDRESS OF MULTIPLIER	ADDRESS OF MULTIPLICAND	ADDRESS OF PRODUCT ROUNDED-OFF		m	u
SPECIAL MULTIPLY	ADDRESS OF MULTIPLIER	ADDRESS OF MULTIPLICAND	ADDRESS OF PRODUCT ROUNDED-OFF	ADDRESS OF NEXT COMMAND	n	v
PRODUCT	ADDRESS OF MULTIPLIER	ADDRESS OF MULTIPLICAND	ADDRESS OF MOST SIGNIFICANT PART OF PRODUCT	ADDRESS OF LEAST SIGNIFICANT PART OF PRODUCT	p	w
COMPARE	ADDRESS OF MINUEND	ADDRESS OF SUBTRAHEND	ADDRESS OF DIFFERENCE*	ADDRESS OF NEXT COMMAND IF DIFFERENCE IS NON-NEGATIVE	c	8
SPECIAL COMPARE	ADDRESS OF MINUEND	ADDRESS OF SUBTRAHEND	ADDRESS OF DIFFERENCE OF ABSOLUTE VALUES	ADDRESS OF NEXT COMMAND IF DIFFERENCE OF ABSOLUTE VALUES IS NON-NEGATIVE	d	9
EXTRACT	ADDRESS OF EXTRACTOR (DETERMINES DIGITS TO BE EXTRACTED)	ADDRESS OF EXTRACTEE	ADDRESS OF EXTRACTED AND SHIFTED RESULT	IF SECOND DIGIT OF δ IS 0-SHIFT LEFT 1-SHIFT RIGHT OTHER DIGITS TELL NUMBER OF PLACES TO SHIFT	e	y
INITIAL INPUT	(INCOMING INFORMATION GOES TO ADDRESS ϵ)			SELECTS INPUT DEVICE AND TYPE OF INPUT	i	0
INPUT	ADDRESS OF INCOMING INFORMATION		CHANNEL ADDRESS ON DRUM	SELECTS INPUT DEVICE AND TYPE OF INPUT	j	1
OUTPUT	ADDRESS OF OUTGOING INFORMATION	ADDRESS OF 1/2 BASE, FOR CONVERTED OUTPUT	CHANNEL ADDRESS ON DRUM	SELECTS OUTPUT DEVICE AND TYPE OF OUTPUT	o	2

* Overflow gives same word in γ as in a and s Commands. ** Equivalent of F symbol in hexadecimal notation.

[The missing symbols are: x = space; z = the letter f, 3 = the letter r]

DEFINITIONS

ARROW (→)	Notation used in remarks column of coding sheets to denote "replaces". "x^2 replaces y^3" is equivalent to "$x^2 \rightarrow y^3$".
CODING SHEETS	(Cf. page II:4)

Cell No.	Card No.	W. P.	S	α	β	γ	δ	F	Cl ε	Remarks

W.P = word pulse, S = sign, Cl ε = clear epsilon
(Cf. page II:2 on clear epsilon)

DUMMY	A constant in storage which resembles a command, but is never obeyed. The routine adds a number to the dummy to produce a command, which is then obeyed. (Cf. page V:1 on modified commands.) The dummy might also be used as a limit against which to compare a command being modified.
ENTRY	A code number, determined by each coder, used by a routine to send control to the beginning of a subroutine.
EPSILON COUNTER	Basic governing unit of SWAC. It specifies the address of the command to be obeyed next. As each command is completed, the epsilon counter normally advances by unity. When epsilon is either cleared by a switch or changed by a special command it continues to count from its new setting. The addition of 255 + 1 in the epsilon counter results in 000.
HALT	There is no halt built into SWAC. Any halt must be coded into a routine. This is most effectively done by addressing a command which calls for input from the SWAC typewriter.
LINK	The address of the command to be obeyed upon completion of a subroutine.
REGISTERS	The M and R registers referred to in the text are transfer or accumulating units of SWAC. They are not part of the memory, but are used in execution of commands.

DEFINITIONS

ROUND-OFF 1	Addition of "1" to the least significant position of the result of the m or n product commands, if the least significant half of the product is "1" in the most significant binary digit. Round-off also occurs in the result of a right shift extract command if the binary digit to the right of the least significant digit in the result was a "1" <u>before</u> shifting.
WORD	The contents of one memory address of SWAC. The 36 binary digits (9 hexadecimal digits) are allotted in the following manner.

	Hexadecimal Digits	Binary Digits
α	1 and 2	1 - 8
β	3 and 4	9 - 16
γ	5 and 6	17 - 24
δ	7 and 8	25 - 32
F	9	33 - 36

(If the word being considered is a command, see pg. iii for meanings of α, β, γ, δ, and F parts of the word.)

I. GENERAL CODING PROCEDURE

Basic Rules

To facilitate checking out routines and operating SWAC, store the following words in cells 00 and 01:

Cell No.	α	β	γ	δ	F	Remarks
00	01	00	00	10	j	Typewriter input to (01)
01	(Temporary; initially zero)					

This device permits the operator to type any desired command into (01) and cause SWAC to obey it immediately. If, for example, the command of 01 01 01 5u t were typed into the machine while epsilon = 00, then the obeying of (01) would send epsilon to (5u) and clear (01).

The cover page of a main routine lists the cells of the memory used by the commands and constants, and the cells assigned to the subroutines. The coder includes instructions for variation of output format if desired. He describes the expected output, whether typed or punched on tape or cards, and the intervals at which it should occur. The operator must know the meaning of various "error" halts and type-outs, what he may inspect to determine the trouble, and where he can enter the routine to re-compute or to continue.

If the coder keeps this last suggestion in mind while coding, he will plan certain "entry points" at which the operator can check coding or perform necessary re-computations. This leads to a strong recommendation for the pre-storing of modifiable commands and variable quantities. Use "work" or "temporary" storage for a variable factor, at the same time leaving the initial value (s) untouched. This enables the operator to re-compute the final value (a) of a cycle repeatedly. When all possibility of error or unexpected overflow is past, code to transfer a new given value to (s) and send the good answer (a) to its proper "work" cell for the next cycle. If one were certain that a cycle of commands were to be performed only once, there would be no harm in destroying factors by re-using their storage cells. However, there may be some trouble in code checking, or some machine trouble, that will make it advisable to re-enter a cycle. The coding sheets should indicate points to which the operator can safely send epsilon to re-compute any portion.

Many add and subtract commands of a routine can never be expected to have overflow. There are others, however, in which overflow may possibly occur, indicating error. The size of a factor may exceed that expected by the coder, or there may be some function computed by a series which causes overflow when the terms are summed. The first suggestion that comes to mind is to let the delta of those commands be 00; then epsilon goes to (00) upon any overflow. However, in this case the operator cannot determine at which command overflow occurred.

If there are enough storage cells available after assigning addresses to the main routine , store several halt commands, 01 00 00 10 j, one for each possible overflow. Code the addresses of the halts in the delta portions of the add and subtract commands which may overflow. In the remarks column of the halt commands indicate which overflow causes each halt; the operator of the SWAC can then type on the output sheet which function had an error, for later interpretation by the coder.

Attached to the coding sheets there must be step-by-step instructions for running the routine. They should describe the amount of memory filled by tape or card initial read-in, the new data to be inserted in the tape reader or collator, the frequency of memory checks, etc.

Example:

1. Read in routine. It goes into addresses 00 to w6; then (00) again; then (zz). It will halt in (00), calling for typewriter input to (01).

   Obey either instruction 2 or 3.

2. IF COMPUTING FROM THE BEGINNING OF PROBLEM:

   Obey (00); type in 01 01 01 uu t.
   Obey (01) and allow machine to continue.

3. IF CONTINUING THE COMPUTING FROM ANOTHER RUN:

   Insert tape of starting values (2 words).
   Obey (00); type in 01 01 01 u8 t.
   Obey (01) and allow machine to continue.

4. TAPE PUNCH-OUTS of results in binary will occur after less than a second of computing:

at epsilon =	address typed out =	factor typed out =	
0x	yu	R_t	
94	yz	c	These 2 are repeated many
95	y5	a	times before a new R_t.

   A total of 50 R_t values should punch out before routine halts in (w7), indicating completion.

5. TYPE-OUTS in groups of six words at epsilon = 51 to 56 indicate a discrepancy in computation; routine will continue. Allow it to continue and save type-out for coder.

6. HALTS

   a. A halt at any epsilon from vx to w6 indicates overflow in the routine. (Please note on type-out sheet the epsilon at which halt or overflow occurred; see coding sheets.) To get back into routine, clear epsilon and obey (00): type 01 01 01 vw t into (01). Obey (01) and allow machine to continue.

   b. A halt at epsilon = uz indicates memory check failure.* Obey instructions on coding sheets at epsilon = uz. Run out some blank tape from punch-out unit. Set machine to command (COM) mode of operation.**

      Obey (uz), i.e., finish the input to (98) = hash.
      Obey (v0) and (v1), tape punch-out of two starting values.

* Cf. page IV:1 for memory check.
** To enable operator to cause execution of one command at a time.

Obey the commands beginning in (v2), memory type-out. The value in (01) is the difference between the given memory sum and the computed memory sum. If this indicates that the memory can easily be fixed, use (00) and (01) to do so.

After fixing the memory obey (00); type 01 01 01 u8 t. Insert tape of two words of starting values just punched out. Obey (01) and allow machine to continue.

If memory error is too bad to be fixed, and routine must be read in again, obey instruction number 3.

Some Aids to Coding

Before beginning the actual coding of a problem, in most instances, draw a flow diagram. This gives a pictorial representation of the steps to be followed in the routine. A sample flow diagram is shown on the following page for the computation of a table of $f(x) = \sqrt{x^2 - 1}$, $x = 1(1)25$. Since square root is not an operation of SWAC it must be computed by a subroutine. However, in the flow diagram the coder treats square root as another operation; details of the process are usually considered in a separate diagram.

Since SWAC has no built-in checks, include coded checks in the routine. In the example given, the result $f(x)$ is squared and compared with $x^2 - 1$. If this agrees within a certain predetermined tolerance, the machine continues. If the check fails, an alternating tally causes the machine to compute $f(x)$ once more. If it again fails, the machine halts.

When doing the initial coding, use the flow chart as a guide to the operations and their sequence. Instead of assigning memory locations to words, use symbols such as R_i for instructions, c_i for constants, and t_i for temporary storage.

Always use a <u>right</u> shift if coding for a zero shift.

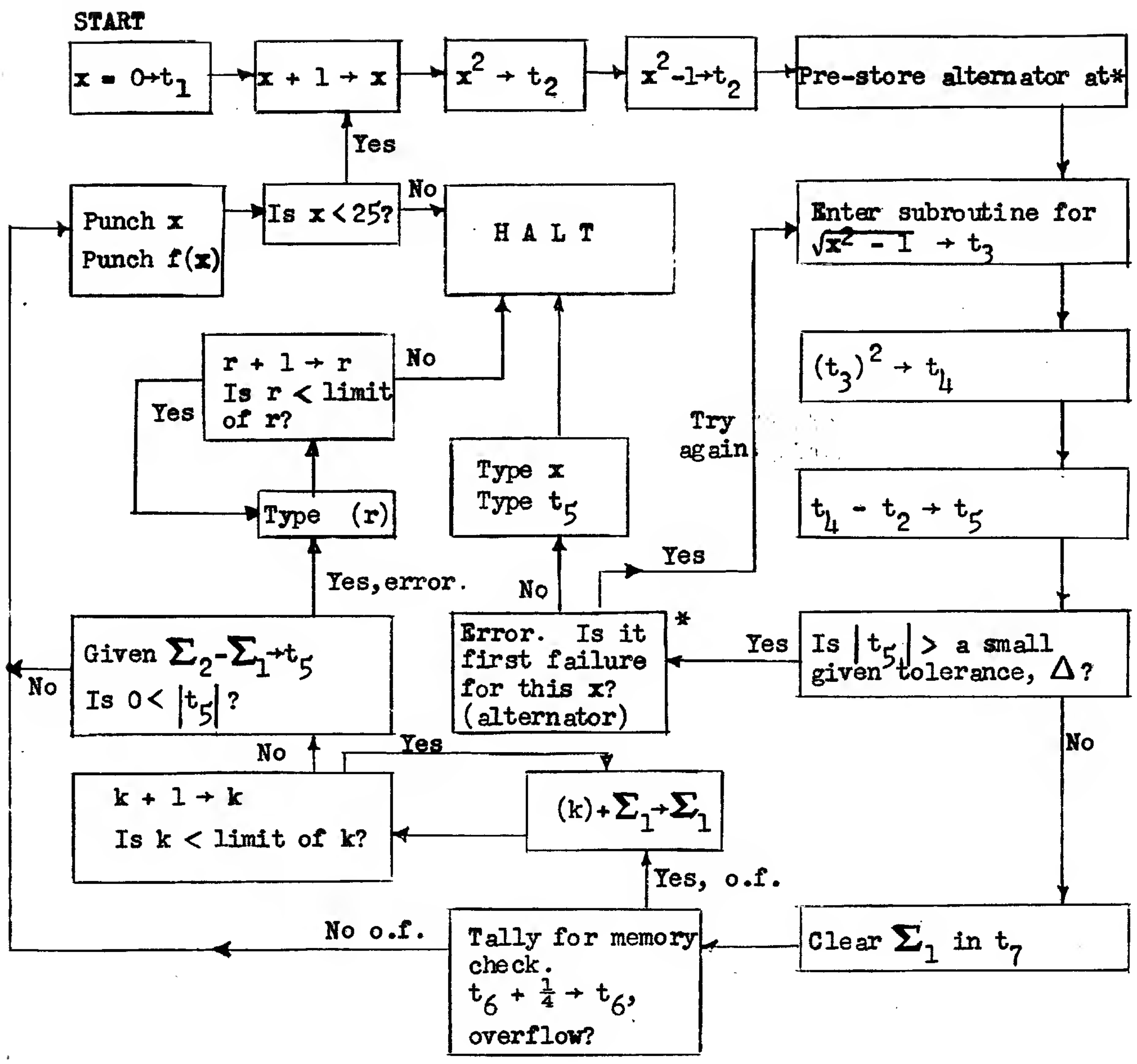

Figure 1. FLOW DIAGRAM FOR COMPUTING $f(x) = \sqrt{x^2 - 1}$, $x = 1(1)25$.

Following is a sample of coding in symbols. This coding covers that portion of the flow diagram which begins with "pre-store alternator at *" and continues through "Clear Σ_1 in t_7". It includes checking the accuracy of the square root, and the starting of the memory check. The sample coding assumes that instruction R_{11} is the last command of the square root subroutine, or is a transfer command placing $\sqrt{x^2 - 1}$ in storage t_3.

Cell No.	α	β	γ	δ	F	Remarks
R_5	t_9	t_9	t_9	R_6	s	Clear alternator in t_9.
R_6	(Entry to square root subroutine)					
⋮						
R_{12}	t_3	t_3	t_4	R_{13}	m	$(t_3)^2 \rightarrow t_4$
R_{13}	t_4	t_2	t_5	R_{14}	s	$t_4 - t_2 \rightarrow t_5$
R_{14}	c_3	t_5	t_{10}	R_{17}	d	Is $\lvert t_5 \rvert > \Delta$?
R_{15}	t_9	c_4	t_9	R_{29}	a	Yes, error. Alternator; first failure?
R_{16}	t_5	t_5	c_o	R_6	t	Yes, first failure. Try again.
R_{17}	t_9	t_9	t_7	R_{18}	s	No error. Enter memory check. Clear Σ_1 in t_7.
⋮	(Remainder of main routine)					
R_{29}	01	00	00	10	j	HALT -- error, or completion of problem.
⋮						
c_o	00	00	00	00	0	Zero storage.
c_1						
c_2						
c_3	00	00	00	00	9	Δ for tolerance on accuracy of sq. rt.
c_4	80	00	00	00	0	2^{-1}
⋮	(Remainder of constants)					
t_7	(Temporary; accumulator of machine memory sum)					
t_9	(Alternator tally; initially zero. Alternately 2^{-1} and zero-with-overflow)					
t_{10}	(Hash)					

After the initial coding is complete, assign memory addresses to the symbols. A formalized method of doing this by IBM machines is described in the memorandum, "Symbolic Coding", by B. F. Handy, National Bureau of Standards, dated September 2, 1952.

When making the assignments, store the constants either directly before or directly after the commands, and assign the temporaries in a group apart from the rest. This group of words may be omitted from the input tape or cards since temporaries need not be read into the machine. (Cf. page II:2 for semicolon) Grouping the temporaries also simplifies the memory check, which is executed at regular intervals (at every fourth x in the given example) to test the constancy of the memory. (Cf. page IV:1 for memory check) Note that the flow diagram contains reminders to the coder to pre-store and pre-clear certain modified commands and temporary storage cells (for example: to clear Σ_1, the cell in which the memory sum is accumulated each time the memory check is executed).

If a temporary storage address, t_i, is used successively for two or more values, the following type of chart keeps track of what is currently stored in t_i. When used during coding, the chart enables the coder to tell whether the value he expects in t_i has been replaced by another value, or has been left intact.

$3y = t_1$	$x(23)$	
$3z = t_2$	$x^2(24)$, $x^2 - 1$ (25)	
$40 = t_3$	$\sqrt{x^2 - 1}$ (26)	[(26) = R_{11}]

The number in parenthesis following each symbol indicates the value of epsilon at the time of transfer or computation. Here, cell t_2 has been used for two results; the coder is reminded that x^2 is not available after epsilon = 25. In this example, commands and constants are stored from 00 to 3x; hence t_1 = 3y.

For the operator of a routine to be able to re-enter the routine, or to change any word in it, the coding must begin with

```
00:  01  00  00  10 j
01:  00  00  00  00 0
```

as mentioned before. To use (00) in this manner, the operator must be able to clear epsilon manually at almost any point in the routine. Keeping this in mind, code with as few "special" commands (those which specify the next epsilon) as possible. The example on the left below makes use of "specials" unnecessarily; it could just as easily be written as on the right.

Cell No.	α	β	γ	δ	F	Remarks
59	69	64	63	5u	b	Transfer (69) to (63).
5u	63	63	62	5v	n	$(63)^2 \rightarrow (62)$
5v	62	63	64	5w	t	$(62)-(63)\rightarrow(64)$
⋮	(Remainder of routine)					
62	(Temporary storage)					
63	(Temporary storage)					
64	00	00	00	00	0	Zero storage.

Cell No.	α	β	γ	δ	F
59	69	64	63	5u	a
5u	63	63	62	5v	m
5v	62	63	64	5w	s
⋮	(Remainder of routine)				
62	(Temporary storage)				
63	(Temporary storage)				
64	00	00	00	00	0

In the routine using the special commands the operator cannot return the control to 00 even if he clears epsilon manually; the presence of a special command dictates the next command because of the delta to epsilon transfer on a special command.

If the data to be introduced in computing has a very wide range, store the numbers in floating binary form, using the floating operations subroutine. This system stores a value as a number q, $\frac{1}{2} \leq q < 1$, accompanied by an appropriate power of 2. Otherwise think of data as numbers less than one, and assign scale factors accordingly. Keep a record of the scale factors assigned to the result of each operation in the remarks column of the coding sheets.

Example:

Compute: $a_i b_i$ where $a_{i+1} = a_i b_i + 7$, and $b_{i+1} = b_i + 2$, until b_i reaches a given limit.

Given: $a_o = 1.69y4$; maximum known to be $a_i = 7.yz49$
$b_o = 4.y496$; maximum known to be $b_i = y.4697$

Store: $2^{-3}a_o = .2x3w80000$ and $2^{-3}a_i = .zxy920000$
$2^{-4}b_o = .4y4960000$ and $2^{-4}b_i = .y46970000$

Cell No.	α	β	γ	δ	F	Remarks
43	56	59	57	44	m	$2^{-7}a_i b_i \rightarrow (57)$
44	81	57	58	04	e	$2^{-3}a_i b_i \rightarrow (58)$
45	58	80	56	46	a	$2^{-3}(a_i b_i + 7) \rightarrow (56)$
46	59	82	59	47	a	$2^{-4}(b_i + 2) \rightarrow (59)$
47	83	59	86	43	c	Is (83) < (59)?
48	(Command to be obeyed after (59) has reached its limit)					
⋮						
56						$2^{-3}a_i$
57						$2^{-7}a_i b_i$
58						$2^{-3}a_i b_i$
59						$2^{-4}b_i$
⋮						
80	y0	00	00	00	0	$2^{-3}7$
81	00	00	00	00	0	Zero storage.
82	20	00	00	00	0	$2^{-4}2$
83	y4	69	60	00	0	Limit of $2^{-4}b_i$.
86	(Temporary storage)					Hash.

II. <u>INPUT</u>

Initial Loading

When a routine tape is to be read into the machine, the operator clears the epsilon counter and the memory storage. The tape is inserted in the tape reader, and the operator presses the OOPB (one-operation push-button) once. One half of a command (or four periods) is obeyed; during this first half the machine inspects the command stored in the memory address designated by epsilon (namely, 00). Since the machine has been cleared, the command stored in (00) is all zeros. An F = 0 signifies "initial input": input to the memory address equal to epsilon.

Then the second half of the command is obeyed. The tape reader advances the tape for one word, and that word reads successively into the R and M registers and into the memory address equal to epsilon. The epsilon counter increases to 01. Both halves of the command take place rapidly in operation on continuous (CONT) or on one command at a time (COM); operation on period (PER) enables one to see each of these transfers separately.

After epsilon advances to (01) the machine inspects the command stored in (01); this, too, is all zeros, and another tape input (to memory address 01) takes place. The SWAC continues to inspect successive commands stored in the memory and to replace them with words from the routine tape as long as it finds both F=0 in the memory and input tape in the tape reader.

When a routine is to be read into the machine from cards, the IBM collator is used instead of the tape reader. The same procedure takes place, however, in the obeying of commands whose "F" is zero. SWAC inspects successive cells in the memory and replaces them with words from cards as long as it finds both F=0 in the memory and cards in the collator.

There is a "stop" character following each word on punched tape.

Corresponding to the "stop" character on tape there is a "word pulse" on cards, one pulse for each word read in. It signals the SWAC to store the contents of a word and to increase epsilon, as does the "stop" on the tape.

Use of the "clear epsilon" character obviates halting the read-in and clearing epsilon manually. Suppose on tape a "semicolon" follows the "stop" character for the last word to be read in (for example, (4u)). The machine will be halfway through the command of initial input into memory address 4v at the time that the "semicolon" character is read from the tape. The "semicolon" code causes epsilon to clear, and the delta of the command currently being obeyed is transferred to epsilon. In the present example, (4v) when inspected by the machine is all zeros; hence, an initial input command (F=0) has been set up to input to the origin.

However, the initial input command has yet to be obeyed. Epsilon now reads 00 instead of 4v, and the next word on the tape will read into memory address (00), clearing whatever may have been there. There may have been a word present on the beginning of the routine tape to be read into memory address (00). It is necessary to repeat (after the "semicolon" character) the contents of (00) followed by a "stop" character in order that (00) may contain the correct word. The machine will inspect (01) for its next order and obey it. The command in (01) <u>may</u> increase $\mathcal{E}$ for additional input.

The "clear epsilon" signal from card input accomplishes the same thing. To code for this on cards place a "5" in the Clear Epsilon column of the coding sheets on the appropriate line. As in tape input, the command in (00) should be on the same line.

<u>General</u>

When F of the command being obeyed is 0, coded as i, the SWAC executes an initial input to the <u>memory</u> <u>address</u> equal to epsilon. This is discussed above in Initial Loading. When F is i, or j, SWAC inputs to the memory address equal to the alpha portion of the command. The delta portion of the input command being obeyed designates the input device called for, regardless of whether F is i or j.

If delta is either 10, 30, or 50, SWAC calls for a typewriter input. This constitutes a halt command, for the machine does not continue its routine until an operator completes the typewriter input. If delta is 70, SWAC calls for a drum input; the drum is not yet in use.

If the delta of an input command is 00, 20, 40, or 60, SWAC calls for either tape or collator (card) input. Only one or the other is to be used at any one time. If the operator has tape ready in the tape reader and also cards in the collator, with collator motor idling, the machine will attempt to input from both devices. This results in the input of nonsensical values. On the cover sheet attached to the coding always make it clear to the operator when to remove tape or cards from input units, or place cards or tape into position for read-in.

Coding Notation for Various Inputs

In coding for tape input use hexadecimal notation (base 16). There are two hex digits each in α, β, γ, and δ, and one in F. The F column may have either the letter or hex representation for operations. (Cf. page iii). The corresponding hex and decimal notations used on SWAC coding sheets for digits are:

DEC:	000	001	002	003	004	005	006	007	008	009
HEX:	00	01	02	03	04	05	06	07	08	09
DEC:	010	011	012	013	014	015	016	017	018	etc.
HEX:	0u	0v	0w	0x	0y	0z	10	11	12	etc.

If coding for card input code in decimal notation; there are three digits each in α, β, γ, and δ, and two in F. (This applies to non-symbolic coding.) In the F column, use the decimal representation for operations. (Cf. page iii and the above table.)

Use hex notation in the Cell Number column of the coding sheets in coding for tape or cards. Card Number column is not used in tape coding. In card coding that column must hold consecutive numbers, for there will be one input card for every ten lines of coding. For instance, corresponding to coding lines 190 through 199 (in decimal), the collator card will be number 19. The Clear Epsilon and Sign columns of the coding sheets are used similarly in hex and decimal coding. The "clear epsilon" code is "5" in decimal notation, and ";" is the corresponding symbol in hex notation. Indicate the sign by "-" or "X". The Word Pulse column is not used at present for either method of coding.

Coding for Data Input

If a routine calls for input of data from tape it calls for one word at a time; the tape advances one word at a time as input commands are obeyed. However, if a routine calls for card input the coding must take into account the number of words on each input card; this may vary from one to ten. The routine must execute the number of input commands corresponding to the number of word pulses on any one card.

There is enough time between the reading of successive lines of each card to allow SWAC to obey two or three commands. To avoid storing ten input commands, the coder may store only one input command. He then must store another command to modify the input command, sending the information from the card to successive memory locations. A tally command is used to control the amount of input, counting to the number of word pulses on each card.. (Cf. page V:1 for modifying commands.)

III. <u>OUTPUT</u>

<u>Type and Tape</u>

When the F of the command being obeyed is 2, coded as o in hex notation, the SWAC executes an output. The alpha portion of an output command determines the cell whose contents will be sent to an output unit. The delta portion of the command determines which output device will receive the information. The beta portion determines the cell whose contents are displayed in the M register on the console during output.

If delta is 00, the SWAC <u>types</u> the contents of the cell referred to in alpha. If delta is 10, there is an output to punched tape. Both of these output devices are governed by an output format. Normally this format has eleven characters:

positive number: space, 9 integers, tab character, "stop"

negative number: minus, 9 integers, tab character, "stop"

The nine digits are typed or punched in <u>reverse</u> order, the most significant at the extreme right.

During typewriter output, the tab character activates the tab key of the SWAC typewriter. The "stop" does not appear on type-out. Both the "stop" and tab characters are punched on output tape. A tape of punched answers can be listed in final form on an auxiliary typewriter, and can also be read back into the SWAC as data for a subsequent routine. The punched tape output is more useful for problems with a small amount of values. Punched <u>card</u> output is much faster, and more useful for voluminous answers.

The tab has no significance if read back into the SWAC from tape; its purpose is to operate the tab key of the auxiliary typewriter, enabling the operator to list answers in columns. The "stop" character merely lists on that typewriter as "/". During read-in to the SWAC it signals the end of one complete word and causes the increase of epsilon.

Either tape or typewriter output can be set by the operator for "converted" output by a switch; it is possible thus to have values expressed to base 10 or 16. With this kind of output the digits are punched or typed in the correct order, the most significant at the extreme left. The operator can also obtain a word of eleven digits in "converted" output. Besides notifying the operator that he desires "converted" output, though, the coder must also code an appropriate cell in the beta of every "00" or "20" type-out. For decimal output, he must have beta of the command refer to a cell containing "5" in the F position; for hex, he refers to a cell containing "8" in the F position. It should be noted that 11 decimal digits are roughly equivalent to 9 hex digits.

If the coder is interested in only part of a word (for example, three hex digits) the operator can cause the tab and "stop" characters to follow directly after the first three integers, thus skipping the other integers. Since a word leaves storage in reverse order during "normal" output, the three integers obtained in this example would be those stored in delta and F. Note that these are the three <u>least</u> significant integers. On "converted" output the digits would be the three <u>most</u> significant integers, two in alpha, and one in beta. The punch or type output of a nine integer word requires two seconds.

It is also possible to change the format of each word. In addition to the integers, tab, and "stop", there are available three space characters and a period. Following the coder's instructions, the operator can intersperse the other characters among the digits of an output word.

Normally the output typewriter is set to type six words per line; if the outputs come in groups of two, three, or six words no change need be made. According to the coder's instructions, the operator can cause the typing of fewer words per line to accommodate groups of four or five words. In all the above variations of output <u>the coder must bear in mind that any change</u> in the output for type or tape will effect <u>all</u> words, whether type or punch tape output; e.g., "converted" output also affects "10" punch-out.

<u>20 Output</u>

If the delta of an output command is 20, SWAC types nine <u>letters</u> representing the nine <u>digits</u> in the cell referred to by alpha. There is a limited alphabet of 15 letters and a space. (Cf. page iii for corresponding digits and letters.) As an example of its use, suppose the coder were interested in knowing only the fact that f(x) had exceeded a limit. He might not have the routine halt in the event of that overflow. However, he might code the routine to type a reminder to re-check the function at x, and then to continue computing with x+1.

The following commands would type such a reminder:

Cell No.	α	β	γ	δ	F	Remarks
32	7x	00	00	20	o	Type "re do".
33	4z	00	00	00	o	Type x at time of overflow.
34	(Continue with rest of computing routine)					
⋮						
4z	Temporary storage assigned to x.					
⋮						
7x	xx	29	xy	3x	x	"Re do" in reverse order, with space characters inserted.

NOTE: The cell 7x could contain xx od xe rx x on the coding sheets and the same letters would be typed from SWAC. (The "x" code is a space.)

If the SWAC has been set for "converted" output, it would be possible to use "20" letter output, providing the beta of the output command refers to a cell with "8" in the F position.

<u>30 Output</u>

If the delta of an output command is 30, nothing takes place. This command is not used, for it causes only a momentary halt in the routine.

<u>40 Output</u>

A delta of 40 in an output command causes SWAC to type one letter or one number representing the last five binary digits in the cell referred to by alpha. The tab and carriage return keys of the output typewriter are not automatically activated as in other type-outs. They must be coded in "alpha" cells just as are the letters. If a routine includes any "40" outputs there <u>must</u> be instructions to the operator to set the "40" output switch either to "numbers" or "letters". (Cf. following page, III:5, for corresponding letters and numbers.) Though only one character is typed for each output command, there is available a complete alphabet.

If SWAC has been set for "converted" output (instead of "normal"), the output typewriter is not activated during "40" output. There is only a momentary halt in the routine.

"40" Output (single character)

If (α) contains:	**SWAC will type**	Switch Setting	
$\underline{\alpha}\,\underline{\beta}\,\underline{\gamma}\,\underline{\delta}\,\underline{F}$		Numbers	Letters
000000000		Nothing	Space
01		Backspace	l
02		Nothing	.
03		Nothing	Carriage Return
04		½	Shift Up
05		'	Shift Down
06		Nothing	g
07		Nothing	h
08		-	tab
09		/	k
0u		,	u
0v		;	v
0w		Nothing	w
0x		Nothing	x
0y		Nothing	y
0z		Nothing	z
10		0	i
11		1	j
12		2	o
13		3	r
14		4	a
15		5	b
16		6	s
17		7	t
18		8	c
19		9	d
1u		u	m
1v		v	n
1w		w	p
1x		x	Space
1y		y	e
1z		z	f

50 Output (See footnote, page III:7)

When the delta of an output command is 50, SWAC causes the number in the cell referred to by alpha to be punched on an IBM card. There must be a group of ten values punched out in succession. The values appear on the card in binary representation, ten to a card. Any of these words punched out may be zero. If a coder has only four values to be punched at each punch-out, six of the words punched will be zero.

The time required for punching of one card allows SWAC to obey two or three commands between the punching of two successive lines (or words). This enables the coder to store only one output command, and employ a cycle of modified commands to transfer the successive output words to the cell in alpha of the output command. (Cf. page V:1 on modified commands.) It saves the storage of ten separate output commands, if extra storage space is needed.

The cards punched from output may be read back into SWAC as data, by the collator. During the punching, a word pulse accompanies each line or word of punching; during subsequent input the word pulse is the signal to SWAC of the completion of one word, causing epsilon to increase. If any of the lines punched out should not be read back into SWAC, the coder must anticipate this at the time of punch-out by requesting the operator to inhibit the word pulse where necessary. Any or all of the word pulse punches may be eliminated.

Routines have been coded which will call for collator input of cards carrying binary values, and punch out cards of the decimal equivalents. This enables a coder to have the punched results of a computation translated to decimal and then listed on IBM equipment. There is also a routine to translate data punched in decimal form to cards in binary notation; the binary data is then ready to be called for as input during a computation routine. Refer to memorandum, "Programs for Conversion of Decimal Punched Cards to Binary Punched Cards", by B. F. Handy, National Bureau of Standards, dated October 14, 1952.

60 Output (See footnote below)

A breakpoint output command is designated by 60 in delta. No output device is activated; breakpoint is merely a halt, accompanied by the display of two values on the console of SWAC. The contents of the cells referred to in the alpha and beta portions of the output command show on the console for comparison with hand-computed values. This is most useful for checking out coding on SWAC; after a routine is considered correct the setting of a switch inhibits the breakpoint halt during actual computation runs. Note that although the breakpoint is not operative with the switch "off", an appreciable delay is introduced as each breakpoint is passed. If there are many breakpoints, an alternate routine, omitting them, should be used during the computation.

70 Output

The output command with 70 in delta calls for an output to the magnetic drum; this is not yet in use.

Tape-to-Card Converter

There is auxiliary equipment which translates punched tape, either in decimal or hex notation, to punched cards; its operation is independent of SWAC. The translation, however, does not change the notation either from decimal to hex or vice versa. This is useful for the comparison of duplicate tape results on two separate computation runs, by using IBM equipment to compare the cards.

Note: Whether SWAC is set for "normal" or "converted" output is immaterial in "50" and "60" outputs.

Summary of Input and Output

Input command: F is 0 or 1 (i or j)

Output command: F is 2 (o)

Delta			
Hex.	Dec.	Input Unit	Output Unit
00	000	Tape or Collator	Typewriter (nine digits)
10	016	Typewriter	Tape Punch
20	032	Tape or Collator	Typewriter (nine letters)
30	048	Typewriter	(not used)
40	064	Tape or Collator	Typewriter (one character)
50	080	Typewriter	IBM Card Punch
60	096	Tape or Collator	Breakpoint
70	112	Drum (not in use)	Drum (not in use)

IV. MEMORY CHECK

Memory Check is a device to notify operator of loss or pickup of numbers in the memory. Summing the memory after routine read-in (or after computation of 1st value) checks immediately for failure to read in a word, or to store it, properly. Intermittent summing in the same manner during computation checks for loss or pickup during operation. One more memory check after the last computation will verify accuracy of the routine during final computations.

Preferably, assign addresses to storage in the following order:

1) Commands.
2) Constants and dummy commands.
3) Temporary storage and (if possible) modified commands.

Assign the last memory location in 2) to the given memory sum. The first time the routine is checked out on the SWAC this sum is entered in the memory as zero. Assign one cell in 3) to the memory sum as computed at regular intervals by the machine.

Whenever a memory check is to be made, the routine enters a short sequence of commands which add 1) and 2) as stored in the memory. The summing routine stops just before the cell containing the given memory sum. It then compares the sum of 1) and 2) as given with the sum computed by the machine and stored in a cell of 3). If they do not agree, indication of memory failure should be given by the machine.

The coder may choose to have the routine type the difference between the two sums and then halt. Possibly an analysis of the difference will indicate the failure. If the difference is large, the only recourse may be to read the routine in again. Hence, it is important to code so that computation may be continued from any point.

The first time the routine is checked out, the memory check causes a failure since the given sum has been entered as zero. The difference between the two sums is in this case the correct sum. On another day the routine is again read into the machine, this time using as the memory sum the type-out obtained on the first trial. If there is no failure, that sum computed by the machine is entered permanently in the routine as the given memory sum, and so used in subsequent computing runs of the routine.

If the given memory sum is stored negatively it can be included in the summing. Then the machine sum should be compared with zero to determine accuracy of the memory. In the event it is not zero, the machine sum should be typed out for analysis of the error.

When a problem requires the use of subroutines the assignment of storage would be the same as described above, with the addition of 4): subroutine commands and constants. In this case, the summing cycle would add the contents of 1) and 2), skip over the contents of 3), and add 4). Then the routine would compare that sum with the given memory sum, as above.

The routine must take into account the few modifiable commands and temporary storage cells which must necessarily be stored in 1), 2), and 4). Since they are not always of the same value, they would always cause the memory sums to disagree. If permissible, those cells should be restored to zero before being included in the machine sum, or they may be omitted. A memory sum should have a constant value; barring that, it should vary in a predictable fashion.

V. MODIFIED COMMANDS

One of the more powerful features of the coding for a machine like SWAC is the possibility of changing commands which are already in the memory. This is called "modifying" and it treats the commands like numbers, operating on them to produce new commands, which can be obeyed in turn. It is often possible to save considerable memory space by ingenious use of modification. The following paragraphs offer some suggestions on modifying commands.

The most straightforward way to code a cycle of, say, six operations that are to be performed eight times is to code in 48 successive cells the eight groups of six commands. However, it may take fewer cells to use the same group of six commands, stored in six cells, eight times over. This requires modifying one or more of the references of the six commands depending on the computation, each time the cycle is performed. It also necessitates a tally count up to eight, and a command to leave the cycle when completed.

In some instances, the amount of shift called for in an extract command varies with the numbers being used. The routine stores a dummy extract command with delta equal to zero or an appropriate constant. This dummy is combined with a computed delta value and used each time the extract is to be obeyed.

Instead of storing thirteen output commands to type out the contents of memory addresses from 49 to 5v, modify one output command and use it repeatedly. Begin with (73) = 49 00 00 00 o. Modify it in alpha, compare it with a limiting dummy, and use it until (5v) has been typed out.

Cell No.	α	β	γ	δ	F	Remarks
73	[49]*	00	00	00	o	(k) out
74	92	73	73	75	a	$k + 1 \rightarrow k$
75	93	73	94	73	c	Through?
76	(Command to be obeyed after (5v) has been typed out)					
92	01	00	00	00	0	2^{-8}
93	5v	00	00	00	o	Limit for (73)
94	(Temporary storage; "waste basket")					

This same method can also be used for a summing routine, such as a memory check.

Another example of a modified command is one used in table look-up. Suppose the logarithms of the integers from 1 to 1x (base 16) were stored in addresses 60 to 7x. A dummy (5z 89 v9 36 a) would be stored in (88). To obtain (log y), y would be added in alpha to the dummy; the resulting command would then be obeyed as in the following example:

To find the logarithm of the integer part of x, number in temporary storage address v5.

Cell No.	α	β	γ	δ	F	Remarks
33	87	v5	v6	04	e	Extract and shift integer part of x to y = alpha of (v6).
34	v6	88	35	35	a	Combine (v6) with dummy and place it in (35).
35	[5z]*	89	v9	36	a	Command to transfer log y to (v9). This command may be put into the memory initially as zero; it is always pre-stored by (34).
36	(Command to be obeyed after obtaining the log y in (v9))					
87	00	0z	zz	zz	z	Extractor.
88	5z	89	v9	36	a	Dummy for (35).
89	00	00	00	00	0	
v5	($2^{-12}x$; known to be $1 \leq x \leq 1x$)					
v6	(Temporary storage; contains $y = 2^{-8}$ times the integer part of x)					
v9	(Temporary storage; contains log y)					

* [ ] = portion to be modified.

Before re-entering a cycle containing a modified command it is necessary to restore that command to its original state. If it could be guaranteed that the command had been modified a given number (n) times, then the amount of modification times n could be subtracted from the command to restore it. However, it is very possible that the modification was done less than n times (for instance, in checking out the routine); therefore, the policy of pre-storing a command <u>before</u> entering a computing cycle will insure its accuracy. (Note that the example of table look-up on the preceding page does not require such pre-storing, but that the type-out example first given requires pre-storing of (73) = 49 00 00 00 o.)

Whenever it is possible in the coding, store a modified command in that portion of the memory designated for temporary storage. This eliminates skipping that address when adding commands and constants for memory check, or subtracting that command from the sum after completion of summing. In the table look-up example, the command obeyed in (35) could be stored and obeyed in (v8) among the temporary storage cells as in the following:

Cell No.	α	β	γ	δ	F	Remarks
33	87	v5	v6	04	e	Int. part of x → alpha of (v6).
34	v6	88	v8'	v8	b	Combine (v6) with dummy and → (v8).
36	(Command to be obeyed after obtaining the log y in (v9))					
87	00	0z	zz	zz	z	Extractor.
88	5z	89	v9	36	b	Dummy for (v8). [(89) is zero]
v5	($2^{-12}x$; known to be $1 \leq x \leq 1x$)					
v6	(Temporary storage; contains $y = 2^{-8}$ times the integer part of x)					
v8	[5z]	89	v9	36	b	Log y → (v9); epsilon returns to 36.
v9	(Temporary storage; contains log y)					

In the type-out example given (top of page V:2), since (73) cannot be a special command (specifying the next epsilon) such storing in the temporary portion of the memory is not possible.

VI. TALLIES

There are three ways of tallying to govern the number of times a particular cycle is followed.

a) A constant may be added to a temporary storage cell until the accumulation causes an overflow. [Example 1]

b) A limiting constant may be compared with unity (the result of the comparison -- or subtraction -- being returned to the constant storage). In this compare tally the constant is reduced by unity; at the same time the direction of routine is governed by the sign of the difference. [Examples 2 and 3]

c) A limiting dummy command (A) may be compared with a modified command (B) until (B) has been modified up to the value of (A). [Example 4]

Tallies to cause one operation (C) to occur alternately with another operation (D) are termed alternating tallies. Either the compare commands [Examples 6 and 7] or the overflow signal of an add command [Example 5] may be used to accomplish this alternation.

Example 1:

Adding until overflow occurs

(As long as (38) does not signal an overflow, the routine will go on to (39). When there is overflow, then routine goes to (4z). If m is properly chosen ($m \geq \frac{1}{n+1}$), (47) is left clear and need not be restored before using this tally again. For example, if n = zz, let (97) = 01 00 00 00 0. If n = z, let m = 10, instead of 01.)

Cell No.	α	β	γ	δ	F
38	97	47	47	4z	a
39	(Command to be obeyed n times)				
47	(Initially zero; successively (in alpha) m, 2m, 3m,..., 0 with overflow signal)				
4z	(Command to be obeyed once, only after (39) has been obeyed n times)				
97	(m)	00	00	00	0

Example 2:

Using compare command

(The results of the compares in (23) will be positive n times; hence, (26) will be obeyed n times before the result of the compare in (23) will become negative. (3u) would have to be reset before using the tally again; it is left with -1 in its storage.)

Cell No.	α	β	γ	δ	F
23	3u	79	3u	26	c
24	(Command to be obeyed once, only after (26) has been obeyed n times)				
26	(Command to be obeyed n times)				
3u	(Initially n; successively n-1, n-2, . . ., 0, -1)				
79	("One" in position corresponding to the units position of n in (3u))				

Example 3:

Using special compare command

(The result of the first compare will be negative and (57) will be obeyed. The next time (56) is obeyed, the absolute value of (67) is used, and the result is again negative. Hence, (57) is obeyed until the compare of (56) results in (67) being 0 -- positive; then (60) is obeyed. Before being used again, the tally in (67) must be restored from 0 to n+1.)

Cell No.	α	β	γ	δ	F
56	v9	67	67	60	d
57	(Command to be obeyed n times)				
60	(Command to be obeyed once, only after (57) has been obeyed n times)				
67	(Initially n+1; successively n, n-1, n-2, . . ., all negative; finally, 0)				
v9	("One" in position corresponding to the units position of n+1 in (67))				

Example 4:

Comparing a modified command

(Before (u5) is obeyed it is modified by (u4) to read 49 in alpha. As long as the alpha of (u5) is ≤ 75, the result of the compare in (u6) is positive, and epsilon returns to (u4). After (76) has been typed out, (u6) has a negative result, and epsilon continues to (u7). Before entering this cycle again, the routine must restore (u5) with alpha = 48.)

Cell No.	α	β	γ	δ	F
u4	u5	wx	u5	u5	a
u5	[48]	00	00	00	o
u6	ux	u5	yz	u4	c
u7	(Command to be obeyed after the memory addresses from 49 through 76 have been typed out)				
ux	75	00	00	00	o
wx	01	00	00	00	0
yz	(Temporary storage; "waste basket")				

Example 5:

Alternating by add command

(Accumulating ½ in alpha of (67) causes alternately no overflow and then overflow. Upon overflow, (67) is left zero; if (24) is obeyed an even number of times, therefore, the cycle can be entered again without resetting (67) to zero.)

Cell No.	α	β	γ	δ	ε
24	67	42	67	49	a
25	(Command to be obeyed the first and all odd times)				
42	80	00	00	00	0
49	(Command to be obeyed the second and all even times)				
67	(Initially zero; alternately 80 00 00 00 0 and zero (with overflow signal))				

Example 6:

Alternating by compare command

(The result of the first compare in (92) is -k in (wx). Since the absolute value of (wx) is used for the second obeying of (92), that result is 0 in (wx). Therefore, the results of the compare alternate from negative to zero (positive). If (92) is obeyed an even number of times, (wx) need not be restored to 0 before re-entering the cycle.)

Cell No.	α	β	γ	δ	ε
92	wx	w4	wx	uv	d
93	(Command to be obeyed the first and all odd times)				
uv	(Command to be obeyed the second and all even times)				
w4	(Any constant, k, used otherwise in routine)				
wx	(Initially zero; alternately -k and zero)				

Example 7:

Alternating of compare command

(Result of the first time obeying (2u) is that (2u) is stored negatively; routine goes to (2v). Since the machine ignores sign in obeying commands, the next time (2u) is obeyed as if it were positive; that result is a +(2u), and epsilon goes to (49). Hence, the results alternate from negative to positive, and (2u) is alternately stored negatively and positively. If obeyed an even number of times, (2u) is left stored positively; it need not be restored before re-entering cycle.)

Cell No.	α	β	γ	δ	ε
2u	60	2u	2u	49	c
	(Initially positive; alternately negative and positive)				
2v	(Command to be obeyed the first and all odd times)				
49	(Command to be obeyed the second and all even times)				
60	00	00	00	00	0

NOTE: Of the three alternator methods of tallying, examples 5 and 6 require five cells of storage each. Example 7, by having (2u) modify itself instead of another cell, requires only four cells.

VII. SUBROUTINES

Not all mathematical operations necessary to complete a problem are available in the thirteen basic commands of SWAC. The most frequently used of these complex operations (logarithm, exponential, trigonometric functions, division, etc.) have been coded as subroutines for inclusion in main routines. There is a short cycle of commands, called the Interpretation Routine, designed for inclusion in any main routine, which provides a standard and uniform method of entering and leaving subroutines.

Regardless of the number of commands (k) required for each subroutine, they have all been coded to be stored in memory addresses from (zz - k + 1) to zz. They are all less than 128 cells in length. If a main routine refers to two or more subroutines, the coder assigns storage to them depending only on the number of cells used by each. (Theoretical example: store the logarithm subroutine from 9u to x3, and the square root subroutine from x4 to y6. However, the coding sheets for these subroutines would be for cells w6 to zz and yx to zz, respectively.)

When the routines are read into SWAC, the subroutines are read in from tape or cards as originally coded; they are read into the memory addresses assigned as in the example above. There is another small routine, the Preparatory Routine, which inspects each word of the subroutines. It modifies the "cross-reference" addresses according to the parameters determined by the coder. (In the logarithm example, the parameter = 2w = zz - x3, the difference between the addresses for which the subroutine was coded and the addresses in which it is to be used. In the square root, that difference = 19 = zz - y6.)

After the main routine and subroutines have been read into the memory, the coding should first cause the Preparatory Routine to modify the subroutines. Some problems require so much storage that the space allotted to the Preparatory Routine must be used for other commands; since the Preparatory Routine is used only at the start, to modify subroutines, no harm is done by storing another subroutine in the addresses formerly occupied by the Preparatory Routine (z2 to zz).

In the example above, using the logarithm and square root subroutines, a third routine could be stored from y7 to zz (for example, the sine subroutine). This routine would also be read into SWAC as coded, but need not be modified as are the two others. After the two subroutines have been modified by the Preparatory Routine, the main routine should enter a cycle of commands which read the third subroutine into addresses y7 to zz from input tape or cards.

Example:

((38) modifies (37) and epsilon goes to 37. When the modification of (37) causes an overflow, epsilon goes to 3u. Overflow occurs only after (37) has caused read-in to (y7) through (zz).)

Cell No.	α	β	γ	δ	F	
37	[y7]	00	00	00	j	
38	37	78	37	3u	a	
39	79	79	79	37	t	
3u	(Command to be obeyed after the last word of the sine subroutine has been read into (zz))					
78	01	00	00	00	0	2^{-8}
79	00	00	00	00	0	Zero storage.

The main routine has a code word stored for each subroutine that is to be entered. The code word is not a command nor a numerical constant, but contains the addresses and parameters needed by the subroutine for computing. For example, the square root code word might contain the following addresses.

Sign	α	β	γ	δ	F
-	Address of n	Address of $\sqrt{n}$	Number of whole integers of $\sqrt{n}$	Address of next command	Number of this code word

Every code word is stored negatively. The Interpretation Routine, mentioned before,. distinguishes code words from commands by sign, and always places the current code word in memory address 16. The Interpretation Routine also extracts the various addresses given in the code word. It uses these addresses to store the pertinent values in certain temporary storage cells. Each subroutine is coded to find the necessary values in those same cells.

In the use of subroutines, the constants needed by the subroutines are stored in specific memory cells, and all subroutines refer to these same memory cells. This group of constants in the assigned positions is called the "pool" of constants.

Memory allocation is as follows:
The cells from 1w through 24 contain specific "pool" constants. Cells from 03 through 09 are set aside for temporary storage used by subroutines. Cells 0u through 1v are occupied by the Interpretation Routine mentioned above. The Preparatory Routine is stored in cells z2 through zz. This leaves cells 25 through z1 for allotment by the coder to his main routine and subroutine storage. By carefully scanning the subroutine coding, the coder may possibly find that not all the "pool" constants and the temporaries in addresses 03 to 09 and 1w to 24 are used by his particular subroutines. In that case, he may assign any unused storage from 03 to 09 and 1w to 24 for constants or storage in the main routine.

The cover page of each subroutine has a description of the operation it performs, a list of the pool and temporary storage cells it uses, the number of cells its commands and constants require for storage, the number of cells to be modified by the Preparatory Routine, and its code word.

This should be all the coder needs to plan his use of subroutines. He need not copy the subroutines to the pages of his coding sheets; an indication of where he plans to store them in the memory is sufficient. In fact, he should avoid copying subroutines as a precaution against copying errors.

There is a library of IBM cards on file, containing the subroutines for card input to the SWAC. By use of the IBM reproducer a coder can obtain a duplicate copy of any subroutine on cards. These he incorporates into the cards containing his main routine.